UNIVERSITY OF GLASGOW

Computing Department

MULTUM OPERATING SYSTEM MEMO NO. 4 (1/11/72)

DRAFT SPECIFICATION OF THE BACKING STORE FILE SYSTEM

1. Introduction.

This document describes the backing store file system in some detail. Section 2 contains a review of the design principles on which these proposals are based. Section 3 contains the specification of the file system, from details of disc pack formats to the associated utility programs. In section 4 two nested subsets of the full system are outlined. It might be useful to consider these

(a) as target specifications for intermediate versions,
(b) as specifications of versions for continued support on small configurations and installations where the full power of the file system is not required.

Section 5 contains some ideas for further extensions to the facilities proposed in the full file system. A glossary of the terms and abbreviations used throughout the document is given as an Appendix.

It is convenient to summarise here the main characteristics of the system being proposed.

(a) An indefinite number of public, private, and non-filestore volumes can be supported.
(b) Multi-file volumes are supported only on direct access devices.
(c) Multi-volume files are supported only on magnetic tape.
(d) Within the limits of feasibility, device independence is achieved.
(e) The file system is "application transparent".
(f) Storage space is allocated dynamically.
(g) A user process can access several files concurrently and a file can be concurrently open to several user processes (in mutually consistent access modes).
(h) Access protection can be applied to (i) the file creator, (ii) users in the same project group (strictly, users charging the account which owns the file), (iii) the user community generally.
(j) Provision is made for "backup" to provide security against loss of data and ease the pressure on online stoarage space.
(k) Comprehensive statistics are collected on the use of files.
(l) Systematic use is made of "redundancy" to improve performance and to ease recovery from failures.
(m) It is easy to specify functional subsets of the total design proposal, and to implement functional extensions.

2. Design Principles.

The following owes a great deal to the work of Stuart E. Madnick in clarifying the structure and function of file systems.

2.1 Uniform File Representation.

The evolution of file systems has passed through 3 phases. In the first, each application contained its own file access routines and assumptions about the physical characteristics and formats of the files were embedded in the logic of the programs. The result was heavy duplication and extreme incompatibility, in both data and programs. See Figure 1.

Figure 1

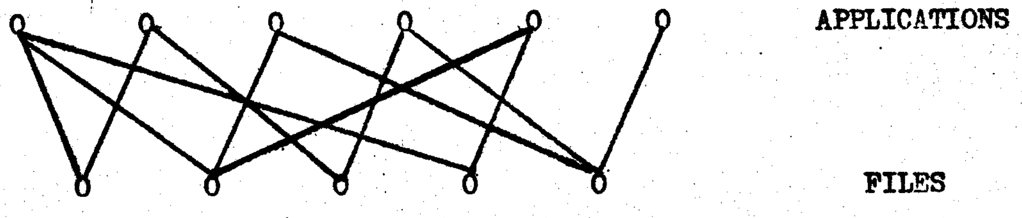


Figure 2

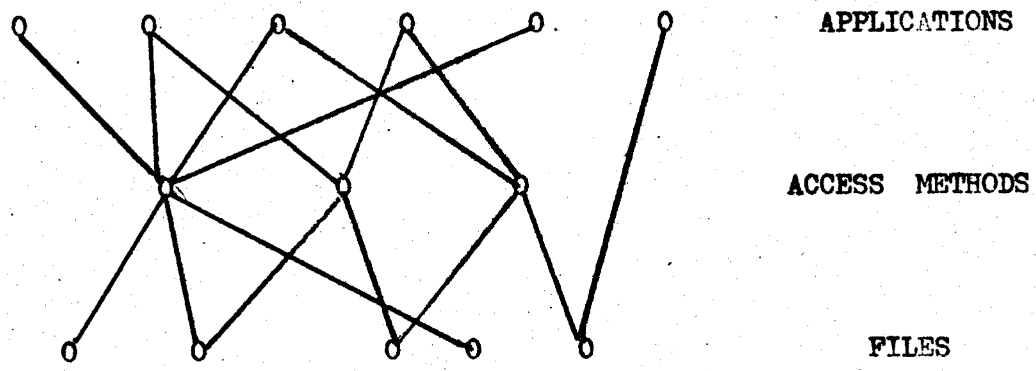
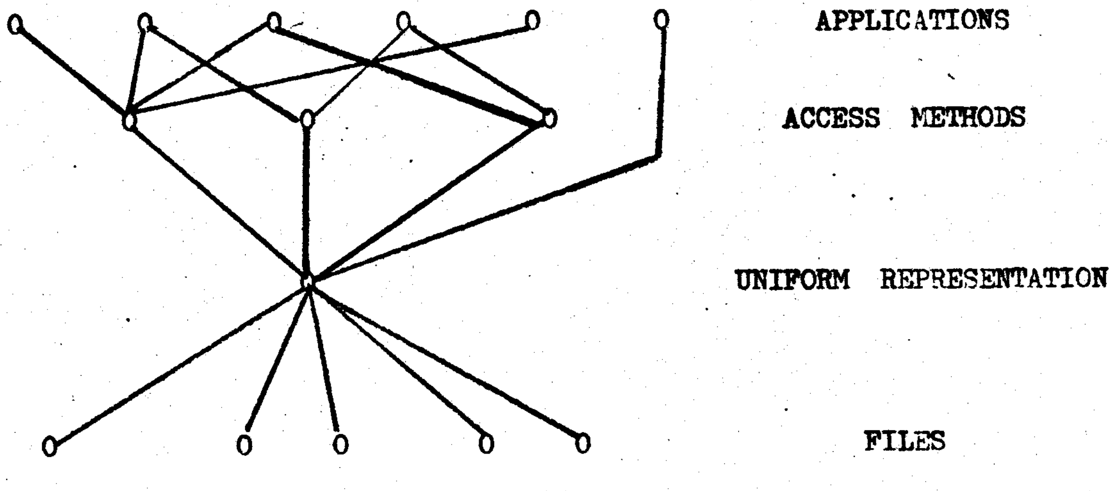


Figure 3



The realisation that a relatively small number of file organisations (or "access methods") are adequate for most purposes resulted in the second phase of file system evolution. Compatibility between programs and data was improved and duplicated coding was much reduced, at the cost of introducing very complex access methods to absorb the varying physical characteristics and formats of the files. See Figure 2.

In the third phase, a "virtual" level with a common file representation was interposed. This resulted in complete program/data compatibility and markedly simplified the access methods. See Figure 3.

This document describes a file system of the latter type. The file system is solely concerned with the physical/virtual transformation and the implementation of the uniform representation. Matters at the logical-record level are the concern of VIOC, access-method packages and applications programs.

2.2 Hierarchical Organisation.

The clearest way of describing a complex system is to define one level of detail solely in terms of the level beneath it. In this way the reader of the description is not confused by irrelevant details and (more importantly) the details at one level may change without affecting the validity of the descriptions at the levels above and below. When applied to software these "levels of detail" are exactly the "levels of abstraction" introduced by Dijkstra.

2.2.1 The File System Hierarchy.

The file system is best explained "top-down", moving from general to the particular, aided by an analogy due to Madnick. Consider the operation of writing to a given location in a given file. The analogy is the operation of sending a letter to "John Smith's home". See Figure 4.

2.2.2 The Filestore Level.

The filestore level (Madnick uses the term "Logical File System") is responsible for converting symbolic file names to more conveniently processable file identifiers. It does this when files are "opened", by consulting the catalog structure. A further task of the filestore level is the protection-checking of "open" requests.

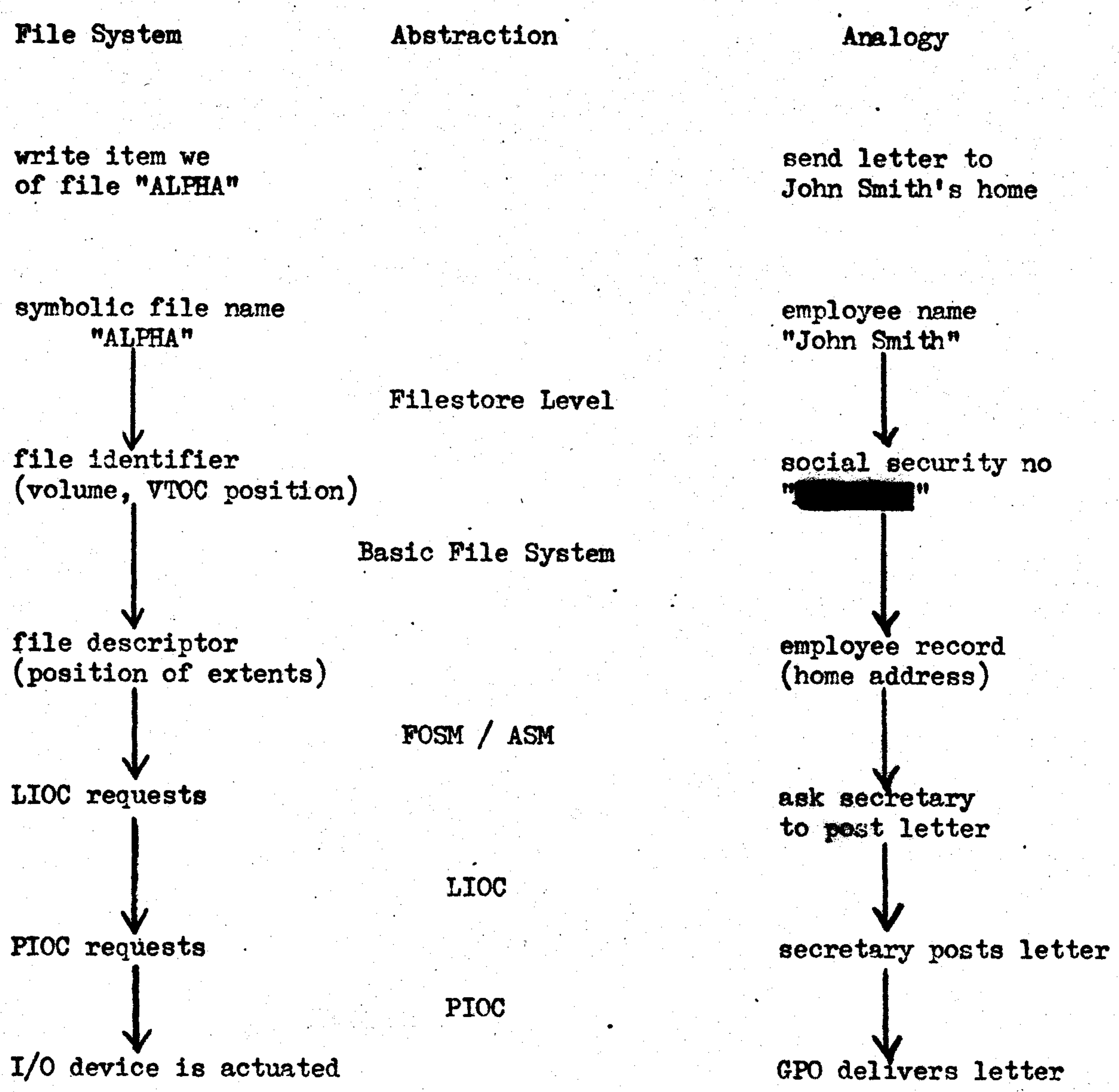
In the analogy, the name "John Smith" is replaced by his social security number, "[redacted]".

2.2.3 The Basic File System.

The basic file system converts file identifiers to file descriptors embodying the physical access data. This is achieved by consulting the Volume Table of Contents (VTOC) for the volume on which the file resides, this being given by the file identifier.

In the analogy "[redacted]" is looked-up in the list of employees and his details, including home address, are extracted.

FIGURE 4


File System
Abstraction
Analogy
write item we of file "ALPHA"
send letter to John Smith's home
symbolic file name "ALPHA"
employee name "John Smith"
Filestore Level
file identifier (volume, VTOC position)
social security no "[redacted]"
Basic File System
file descriptor (position of extents)
employee record (home address)
FOSM / ASM
LIOC requests
ask secretary to post letter
LIOC
PIOC requests
secretary posts letter
PIOC
I/O device is actuated
GPO delivers letter

2.2.4 The File Organisation Strategy Modules.

The FOSM level is responsible for issuing a series of LIOC requests to implement the operation required, taking the access data from the file descriptor.

In the analogy, one instructs one's secretary to send the letter to the given address.

The rest of the task is accomplished by LIOC (The secretary posts the letter) and by PIOC(the GPO flies the letter to London and a postman delivers it to John Smith's house).

2.2.5 The Allocation Strategy Modules.

The ASM level is responsible for freeing unused storage space and claiming free storage space to allocate to files. It is often convenient to consider this level in combination with the FOSM.

In terms of our analogy, the ASM level represents the estate agent who found John Smith his house when he moved to London from Auchenshuggle.

3. A Multum Implementation.

This section outlines an implementation of the design principles for the Multum. Inessential detail has been omitted where it would obscure the broad outline and where it has not yet been defined.

3.1 Volume Formats and Storage Structures.

The file system caters for 3 kinds of direct-access volume. If a non-filestore volume is mounted on a drive allocated to the file system, it must be recognised as such and rejected. Private filestore volumes are managed by the file system on behalf of a designated user, and no other user can claim storage space on them. Public filestore volumes are managed by the file system on behalf of the user community as a whole. To assure the appropriate action from the file system a minimal common format must be enforced for all volumes. This consists of a single sector dedicated to the function of volume identification.

3.1.1 Volume Identification Sector.

The volume identification sector (VIS) is located at a fixed position on each volume and must be free of flaws if the volume is to be usable by the file system. The VIS contains

(a) the "direct access volume label", DVL, an alphanumeric volume identifier which includes an installation identification field;
(b) a status code to indicate whether the volume is public, private, or non-filestore (in the last case only a subset of the remaining fields of the VIS is significant);
(c) the size of the basic allocation unit for the volume (see 3.1.2);
(d) the size of the small allocation unit (see 3.1.2);
(e) the maximum size of the small regime (see 3.1.2);
(f) the size of the large allocation unit (see 3.1.2);
(g) the maximum size of the large regime (see 3.1.2);
(h) a pointer to the store image library area (see 3.1.3);

(j) a pointer to the reserved program library area (see 3.1.4);
(k) a pointer to the volume table of contents (see 3.1.5);
(l) a list of flawed extents (where an unrecoverable parity error has developed), beginning with a word giving the number of entries in the list.

The volume becomes unusable to the file system if more extents become flawed than can be described in the VIS. In the event of this happening, or of the VIS itself failing, a recovery program must be run to salvage any unique data held on the volume. A maximum of about 50 flawed extents can be tolerated.

3.1.2 Allocation Units and the Volume Allocation Map.

On each volume, space is administered in quantities which are an integral number of basic allocation units. The basic allocation unit for a given volume is fixed by its device type and will always be a sub-multiple of the track size. To reduce the overheads which would arise without the facility, two "regimes" of allocation strategy can be defined for each volume and each regime will have its own allocation unit. For convenience, we will refer to these as the small allocation unit and the large allocation unit (SAU and LAU respectively). Both the SAU and the LAU must be multiples of the basic allocation unit (BAU). SAUs are allocated from one end of a volume (e.g. track 0) and LAUs are allocated from the other end. The <u>maximum</u> size of each regime is held in the VIS, along with the sizes of the SAU and LAU. Note that either regime could be given a zero size, and that their combined maximum sizes could be chosen to exceed the capacity of the volume (this gives a "floating" partition).

The standard policy for using the regimes is to choose the LAU much larger than the SAU. The LAU regime would then be used for very large, stable files, while the SAU regime would be used for the smaller, more volatile files. Many other policies are possible and the installation is free to determine these, for each volume it initiates to the filestore, by varying the parameters recorded in the VIS.

The volume allocation map (VAM) is a bit-table with 2 bits for each BAU in the volume, recording the allocation status of that BAU. Three possibilities exist:

(a) a BAU can be free for re-use,
(b) a BAU can be allocated to a file,
(c) a BAU can be neither free nor allocated.

(The last case arises only when a BAU was allocated to a file which has been deleted from the volume since the most recent file system checkpoint.)

In fact, the VAM is held as two parts, each with one bit per BAU. The first part, VAM1, indicates whether each BAU is free, and is used to search for space when allocating storage to a file. The second part, VAM2, is used only to indicate whether non-free BAUs are actually allocated, or merely "pending deletion".

The VAM is an example of "planned redundancy", since it can be reconstituted from the contents of the VIS and the extent tables in the VTOC. (This is actually done after a major file system failure.) The value of the VAM lies in the major simplification and speed-up it permits in the Allocation Strategy Modules and the consistency checks it affords on file system restart.

3.1.3 The Store Image Library Area.

The Store Image Library (SIL) Area is allocated only on volumes used as "swapping" devices. It holds copies of currently "active" segments which have been dumped from main store. The SIL is organised, internally, by the Space Administration procedures of the Executive and is not considered further here.

3.1.4 The Reserved Program Library Area.

The Reserved Program Library (RPL) Area need be allocated only on volumes used as "bootstrap" devices. It contains copies of the bootstrap and nucleus procedures of the Executive, the File System, the Operating System(s) and, perhaps, special real-time programs. The RPL is administered by the system generation and restart routines and is not considered further here.

3.1.5 The Volume Table of Contents.

The Volume Table of Contents (VTOC) is allocated on all filestore volumes. The VTOC contains the file descriptors of all files resident on the volume. In all filestore volumes the first few VTOC entries describe the VIS and VAM, the SIL, the RPL and the VTOC itself. By this means all data on a volume, including the file system's administrative data, can be accessed via the file system. The VTOC structure will now be considered in more detail.

Each entry in the VTOC occupies 128 bytes, half a sector, and contains the following data:

(a) the local name of the file;
(b) the identifier of the parent catalog;
(c) the class of the file(temporary, archival, permanent, immobile or magtape);
(d) the expiry date;
(e) the maximum and current sizes of the file;
(f) a list of the extents occupied by the file, to a maximum of 16 extents;
(g) a field of 16 bytes for use by VIOC and the access method packages;
(h) a field reserved for extensions of the file system.

The extent list consists of pairs of integers. The first number of each pair gives the virtual file address of the last BAU included in the extent, while the second number gives the physical position of the first BAU in the extent. Although a maximum of 16 extents is allowed for, it is inadvisable to allow the storage to become so fragmented that files of about 10 or more extents become numerous, and the volume reorganisation utility should be run often enough to avoid this situation (see 3.3.3). Allowing the storage to fragment badly causes an increase in access time, and increases the probability that a file extension will fail because there is not sufficient contiguous free storage.

3.1.6 The System File Catalog.

The structure of the filestore is that of a rooted tree, the root being the system file catalog (SFC). The SFC is the parent of a number of system data files and also of the user file catalogs (UFCs). Both UFCs and the SFC contain one entry per subordinate file, each entry holding the following data:

(a) the local name of the file;
(b) the account to be charged for the file;
(c) the access permission codes of the creator, his partners, and the public;
(d) the name of the creator;
(e) the date on which the file was created and the date after which it expires;
(f) statistics giving the number of times the file has been opened (for each of the access modes) and the number of times each operation has been performed (averaged over the more recent openings);
(g) the class of the file;
(h) for magnetic tape files only - a list of the reels comprising the file;
(j) for all other files - the file identifier, allocation unit size, backup status, dump file name and dump file position;
(k) the maximum number of generations of the file to be kept by the system;
(l) a pointer to the previous generation of the file (if such exists);
(m) a field reserved for extensions to the file system.

Each catalog entry occupies 128 bytes, half a sector.

3.1.7 The User File Catalogs.

Each user of the file system is associated with one UFC, which acts as the root of his subtree of the filestore. Note that two or more users might share a UFC, their files being distinguished by the field in the catalog entry which gives the name of a file's creator.

3.1.8 USERLIST - A System File.

The Operating System maintains a file called USERLIST which contains the profile of all authorised users of the system. Two components of a user's profile are of particular interest to the file system:

(a) one field gives the identifier of his UFC;
(b) the other field lists the accounts to which he may charge his work (including the file storage he reserves).

3.2 File System Services.

The services offered by the file system are invoked by means of Executive calls which effectively send a message to the file system. This indirect method has been adopted to avoid the need for the user to know the process number of the file system. Details of the formats of these Executive calls are provided in Appendix 2. The following sections concentrate on the nature and scope of the services offered.

Note that many of these services are best regarded as "primitives" from which more useful operations can be built. In the context of the Operating System, several more powerful file operations are available as job control commands. However these are all implemented in terms of the facilties described here.

3.2.1 Access Modes and Protection.

All users of the file system can be classified into one of 3 groups relative to a given file. These groups are:

(a) the (unique) creator of the file;
(b) the (possibly empty) set of partners of the creator;
(c) the rest of the users - the public.

The catalog entry of a file specifies a code for each group, giving the modes in which a user in that group may access the file. The code may indicate permission to access the file in any, all or none of the following 6 modes:

(a) LOAD mode permits the System Loader to read the file in order to establish a process on behalf of the user;
(b) READ mode permits the file to be read without restriction;
(c) APPEND mode permits reading before the current end of the file and writing beyond that point;
(d) UPDATE mode permits the file to be read and written between the execution of the "lock" and "unlock" primitives;
(e) WRITE mode permits unrestricted writing to the file;
(f) CHANGE mode permits deletion of the file and alterations to many of its attributes (e.g. local name, expiry date, protection codes).

A user is permitted to open a file in a given mode only if the relevant protection code includes that mode, and he is permitted to perform an operation on a file only if it is open to him in a mode which allows the operation. The creator of a file is considered to have CHANGE access to it at all times (but he is allowed to alter an attribute or delete the file only if he has opened it first in CHANGE mode).

3.2.2 Creating and Naming.

When a user creates a file he gives it a local name - the name which appears in the file's catalog entry. A local name

name has two components:

(a) the first component, the short name, is a 12 character alphanumeric string (left justified and padded with NULLs when fewer than 12 alphanumerics are used);
(b) the second component, the file type, is a 4 character alphanumeric string (with similar conventions).

The 16 characters taken together must not duplicate the local name of any of the user's files already present in the catalog. The short name, which must begin with a letter, need not be unique as the files can then be distinguished on the basis of the file type. File types are provided mainly as a user facility, and are interpreted by the file system only to the extent that some strings are reserved and not generally available to users.

The creator of a file can always refer unambiguously to it using just its local name (or its short name if this is unique). Other users must prefix it either:

(a) by the creator's name, followed by an apostrophe (e.g. JONES' FILE2); or
(b) by the file name of the catalog containing the file, followed by a colon (e.g. CAT29:FILE2).

When the file type is present in the local name it is written after the short name, and separated from it by a solidus (e.g. FILE1/FORT or FILE1/OCL).

The full name of a file is its (optionally creator-qualified or catalog-qualified) local name, optionally followed by its relative generation number in parentheses. A relative generation number is a negative integer such that, e.g. - 1 is the "father", - 2 the "grandfather" and so on. Thus any of the following are valid full names of a file:

(a) MYFILE
(b) MYSOURCE/ALGL
(c) SMITH'TEXT
(d) SMITH'TEXT/URDU
(e) SJBCAT:TEXT
(f) SJBCAT:TEXT/URDU
(g) PAYMASTER
(h) PAYMASTER (-1)
(j) MYSOURCE/ALGL (-2)

The CREATE operation takes the local name of the file as its first parameter. Additional parameters specify:

(a) the retention period, in days;
(b) the file class (one of magtape, temporary, archival, permanent, immobile);
(c) the number of generations of the file to be maintained;
(d) for magnetic tape files - a field indicating the maximum number of reels allocated to any generation of the file;
(e) for all other files - a field indicating the maximum size of any generation (expressed in sectors as a 32 bit integer) and a flag to indicate whether the small regime is preferred;

(f) the DVL of the volume on which the file should be allocated (this field is obligatory if the file is to be placed on a private volume and optional otherwise);
(g) for immobile files - the number of BAU at which the first extent should start.

If all is well, CREATE will set up a catalog entry for the file and also (for non-tape files) an entry in the chosen volume's VTOC. No space will have been allocated to the file. Initially, the protection status will allow free access to the creator and no access to partners or the public. The file will be left open to the user in CHANGE mode.

See Appendix 2 for a fuller specification of CREATE, including the action taken on failures.